CrowdScreen: Algorithms for Filtering Data with Humans

Aditya Parameswaran
Stanford University
adityagp@cs.stanford.edu

Hector Garcia-Molina
Stanford University
hector@cs.stanford.edu

Hyunjung Park
Stanford University
hyunjung@stanford.edu

Neoklis Polyzotis
UC Santa Cruz
alkis@cs.ucsc.edu

Aditya Ramesh
Stanford University
aramesh1@stanford.edu

Jennifer Widom
Stanford University
widom@cs.stanford.edu

ABSTRACT

Given a set of data items, we consider the problem of *filtering* them based on a set of properties that can be verified by humans. This problem is commonplace in crowdsourcing applications, and yet, to our knowledge, no one has considered the formal optimization of this problem. (Typical solutions use heuristics to solve the problem.) We formally state a few different variants of this problem. We develop deterministic and probabilistic algorithms to optimize the expected cost (i.e., number of questions) and expected error. We experimentally show that our algorithms provide definite gains with respect to other strategies. Our algorithms can be applied in a variety of crowdsourcing scenarios and can form an integral part of any query processor that uses human computation.

1. INTRODUCTION

Crowdsourcing enables programmers to write procedures that employ Human Computation [14] at a large scale in order to solve problems that are hard to solve algorithmically, such as understanding text, sound, video and images.

Recently, there has been a lot of interest in the database research community in using crowdsourcing within a database [4, 7, 8, 12, 13]. The vision is to be able to pose declarative queries that not only refer to stored data, but also to data computed on-demand from humans. The goal is to design a query processor and optimizer that automatically decomposes the query into small unit tasks that are answerable by humans. These unit tasks could be as simple as comparing two items, rating an item or answering a Boolean question.

As part of the query processor, we require human-computation versions of algorithmic building blocks, such as sorting, filtering, searching, categorization or clustering, where the basic operations are done by humans. For instance, humans may be asked to sort a set of profile images by some desired property (such as beauty or age), or to categorize a data set of videos into humor, action, drama and so on. The implementation of these building blocks poses a new set of challenges, as it has to take into account mistakes in human input, monetary compensation for humans, and the inherently high latency of human computation.

In this paper, we focus on one of these fundamental building blocks, an algorithm to filter a set of data items. For example, we may have a set of images, videos, or text, and we want to use humans to decide if the items have particular properties. We use the term *filter* for each of the properties we wish to check. For instance, one filter could be "image shows a scientist," and another could be "people in image are looking at camera." If we apply both filters, we should obtain images of scientists looking at the camera. The emphasis in this paper is on applying a single filter, however, we also provide extensions to conjunctions of filters.

At first sight, the problem of checking which of a set of items satisfy a filter seems trivial: Simply take each item in turn, and ask a human a *question*: Does this item satisfy the filter? The solution is the subset of items that received a positive answer. Unfortunately, humans may make mistakes, so we may not get a desirable solution with such a simple strategy. We may instead show each item to multiple people and somehow combine the answers, e.g., if 2 out of 3 vote yes, then we say the item satisfies the filter. But what is the right way to combine the votes? And how many questions should we ask per item? Does it matter if humans are more likely to make false positive mistakes, as opposed to false negatives? And what if we know a priori that most items are very likely to satisfy the filter? How does this knowledge change our filtering approach? Furthermore, we may not have an unlimited budget or time to do our filtering. So how do we select a strategy that, for instance, minimizes costs (e.g., total number of questions asked) while keeping overall error below a desired threshold?

The problem of filtering with humans is, well, as old as humanity itself, so not surprisingly there is a lot of prior work. Our work is reminiscent of statistical hypothesis testing [19]. While the underlying model is similar to ours (our hypothesis is that an item satisfies a filter), the fundamental difference with our work is in the optimization criteria: With hypothesis testing, one wishes to ensure that the decision on each item meets an error or cost bound. In our case, on the other hand, we filter a large set of items, and our bounds are on the *overall* error or cost. In Section 2 we argue that these optimization criteria are often desirable in crowdsourcing. Furthermore, in Section 7 we show that use of our criteria can lead to substantial cost savings over the traditional, more conservative approach.

Emerging crowdsourcing applications [4, 5] have implemented various types of filtering (e.g., majority voting), but have not studied how to implement optimal filtering strategies. Prior work on using human feedback for information integration [9] (which we will show is worse than our algorithms) uses a heuristic approach to solve the problem. Previous work has also looked at the problem of deciding how and when to obtain labels for machine learning [17, 11, 10], but not the problem of minimizing cost. We revisit related work in more detail in Section 8.

Note that in addition to being useful for query optimization, filtering is also of independent interest in human computation. For example, detecting spam websites in a set of websites (a very common task on Mechanical Turk [1]) is also an instance of the single-filter problem. Relevance judgments for search results [2], where the task is to determine whether a web-page is relevant or not for a given web-search query, is another instance.

Our contributions are the following:

- We identify the interesting dimensions of the single filter problem, which in turn reveal which parameters of the problem can be constrained or optimized in a meaningful manner in Section 2. We formulate five different versions of the problem under constraints on these parameters. We also develop a novel grid-based visualization to reason about strategies for the problems that we describe.
- For deterministic strategies, in Section 3, we develop an algorithm that designs an approximately optimal deterministic strategy (for the cases where we definitely want a deterministic strategy due to simplicity of representation or not having to toss a random dice).
- For probabilistic strategies, in Section 4, we develop a linear programming solution to produce the optimal probabilistic strategy. Our approach naturally generalizes to a setting where multiple possible answers may arise, instead of just two, for instance, colors: red, blue, green, and so on.
- We discuss how our techniques can be extended to the case of multiple filters in Section 6.
- We show that our algorithms perform exceedingly well compared to standard statistics approaches as well as other naive and heuristic approaches in Section 7.

2. PRELIMINARIES

For the majority of this paper, we consider the single filter problem, but we do generalize to multiple filters in Section 6. Also, we assume that our filters are "binary" (i.e., they simply return YES or NO), rather than the "n-ary" filtering case, where the filters may return one out of a set of n disjoint alternatives. (This "n-ary" problem is relevant when we are classifiying each item into exactly one out of a set of classes, e.g., assigning colors.) Our techniques generalize directly to the "n-ary" filtering problem.

2.1 Formal Definitions

We are given a set of items $\mathcal{D}$, where $|\mathcal{D}| = n$. We introduce a random variable V that controls whether an input item satisfies the filter ($V = 1$) or not ($V = 0$). The selectivity of our filter, s, gives us the probability that $V = 1$ (over all possible items). The selectivity may be estimated by sampling on a small number of items. (The sampling approach is commonly used by query optimizers to estimate the size of a selection operator.)

However, we assume we cannot examine an item and determine with certainty whether that item satisfies the filter or not. The only type of action we can perform on an item is to ask a human a *question*. The human can tell us YES (meaning that he thinks the item satisfies the filter) or NO. The human can make mistakes, and in particular:

- The false positive rate is:

$$Pr[\text{answer is YES}|V = 0] = e_0$$

- The false negative rate is

$$Pr[\text{answer is NO}|V = 1] = e_1$$

(We use two distinct probabilities of error because our experience with Mechanical Turk indicates that the false positive rate and false negative rate for a filter can be very different.) We can ask different humans the same question to get better accuracy, and we assume that their errors are independent. (Thus we assume all humans are able to answer the question with the same degree of accuracy.)

A *strategy* is a computer algorithm that takes as input any one item, asks one or more humans questions on that same item, and eventually outputs either "Pass" or "Fail". A Pass output represents a belief that the item satisfies the filter, while Fail represents the opposite. Of course, a strategy can also make mistakes, and our job will be to design good strategies, i.e., ones that "make few mistakes without asking too many questions". We will express this goal more formally later on.

A strategy can be visualized by a two-dimensional grid like the one in Figure 1(a). The Y axis represents the number of YES answers obtained so far from humans, while the X axis is the number of NO answers so far. A grid point at (x, y) determines what the strategy does after x NOs and y YESs have been received from humans: A blue grid point indicates that the strategy outputs Pass at this point, while a red point indicates a Fail decision. We call a point that is either blue or red a *termination point*. At a green point no decision is made and the strategy issues another question, and thus moves to either $(x, y + 1)$ (if an additional YES is received) or to $(x+1, y)$ (if a NO is received). We call green points *continue points* (i.e., we continue to ask questions.) Thus, the evaluation of an item starts at $(0, 0)$ (no questions have been asked), and moves through the grid until hitting either a blue or red grid point. Note that the black points are not reachable under any circumstances. Our example in Figure 1(a) depicts a strategy that always asks a fixed number of questions, in this case 4. Thus, the termination points are along the $x + y = 4$ line.

Although we have described the strategy of Figure 1(a) as processing a *sequence* of YES/NO answers, it can also represent a scenario where a batch of questions is asked at once. (In some systems such as Mechanical Turk [1] it is more effective to issue batches of questions.) If a batch of questions is issued, the strategy describes what to do as each answer is received and processed. If a decision is reached before all answers are received, the outstanding questions can be canceled. If we process all answers and need more, we can issue another batch of questions. (Note that we do not need to issue another batch if we ask the maximum number of questions needed by the strategy in the first batch itself.)

As a second example of a strategy, consider Figure 1(b). In this strategy, we stop as soon as we get four YESs or four NOs. Thus, the total number of questions will vary between 4 and 7.

The astute reader will notice that both of our sample strategies have a certain structure: The termination points (blue/red) form an uninterrupted path, starting at the y axis and ending at the x axis. All the points between this path and the origin are green, and all points outside this area are white (cannot be reached). In Section 3.2 we will study strategies with these properties.

Also note that the strategies of Figure 1(a) and 1(b) are *deterministic*, i.e., the output is the same for the same sequence of answers. In Section 4 we discuss *probabilistic* strategies where the decision at each grid point is probabilistic. For instance, at a particular grid point we may continue asking questions with probability 0.6, may determine Pass with probability 0.3 and Fail with probability 0.1. Thus we represent each grid point by a triple like $(0.6, 0.3, 0.1)$. For deterministic strategies, the only triple values allowed are $(1, 0, 0)$ (green point), $(0, 1, 0)$ (blue point) or $(0, 0, 1)$ (red point).

Note that our strategies are defined to be naturally *uniform*, i.e.,

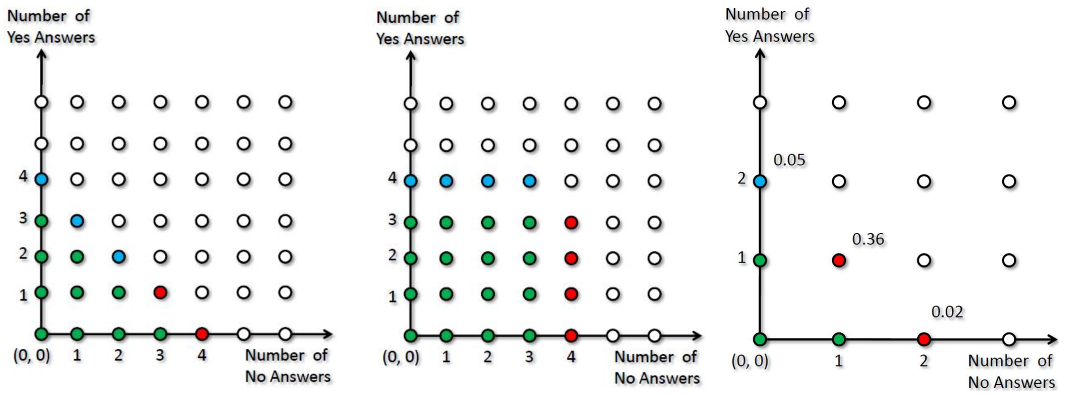


Figure 1: (a) Visualization of a Triangular Strategy (b) Visualization of a Rectangular Strategy (c) Error at Terminating Points vs. Overall Error

the same strategy is applied to each item in the data set, and *complete*, i.e., the strategy tells us what to do at each reachable point in the grid.

In addition, we want our strategies to be *terminating*, i.e., the strategy always terminates in a finite number of steps, no matter what sequence of YES/NO answers are received to the questions (a terminating strategy effectively corresponds to a closed shape around the origin). Note that the strategies in Figures 1(a) and 1(b) are terminating. We will enforce a termination constraint in our problem formulations in Section 2.3.

We also want our strategies to be *fully determined in advance*, so that we do not need to do any computation on the fly while the answers are received. Thus, we present algorithms that compute complete strategies.

2.2 Metrics

To determine which strategy is best, we study two types of metrics, involving error and cost. We start by defining two quantities, given a strategy:

- $p_1(x, y)$ is the probability that the strategy reaches point (x, y) and the item satisfies the filter ($V = 1$); and
- $p_0(x, y)$ is the probability that the strategy reaches point (x, y) and the item does not satisfy the filter ($V = 0$).

Below we give a simple example to illustrate these quantities, and in Section 2.4 we show how to compute them in general. Furthermore, let us say if a Pass decision is made at (x, y) then $\mathsf{Pass}(x, y)$ holds, and that if a Fail decision is made, then $\mathsf{Fail}(x, y)$ holds. If either decision is made, we say that $\mathsf{Term}(x, y)$ holds.

We can now define the following metrics:

- $E(x, y)$ (only of interest when $\mathsf{Term}(x, y)$ holds) is the probability of error given that the strategy terminated at (x, y). If $\mathsf{Pass}(x, y)$, then an error is made if $V = 0$, so $E(x, y)$ is $p_0(x, y)$ divided by the probability that the strategy reached (x, y) (i.e., divided by $p_0(x, y) + p_1(x, y)$). The $\mathsf{Fail}(x, y)$ case is analogous, so we get:

$$E(x, y) = \begin{cases} \frac{p_0(x,y)}{[p_0(x,y)+p_1(x,y)]} & \text{if } \mathsf{Pass}(x, y) \\ \frac{p_1(x,y)}{[p_0(x,y)+p_1(x,y)]} & \text{if } \mathsf{Fail}(x, y) \\ 0 & \text{else} \end{cases} \quad (1)$$

- E is the expected error across all termination points. That is:

$$E = \sum_{(x,y)} E(x, y) \times [p_0(x, y) + p_1(x, y)]. \quad (2)$$

- $C(x, y)$ (only of interest when $\mathsf{Term}(x, y)$ holds) is the number of questions used to reach a decision at (x, y), i.e., simply $x + y$. We consider $C(x, y)$ to be zero at all non-termination points (where $\mathsf{Term}(x, y)$ does not hold).
- C is the expected cost across all termination points, i.e.,:

$$C = \sum_{(x,y)} C(x, y) \times [p_0(x, y) + p_1(x, y)] \quad (3)$$

Note that to evaluate our n items using the same strategy, we will incur an expected cost of nC.

To illustrate these metrics, consider the simple deterministic strategy in Figure 1(c), and assume that $s = 0.5$, $e_1 = 0.1$, $e_0 = 0.2$. At each termination point (x, y) we show $E(x, y)$. For example, $E(0, 2) = 0.05$. This number can be interpreted as follows: In 5% of the cases where we end up terminating at $(0, 2)$, an error will be made. In the remaining 95% of the cases a correct (Pass) decision will be made. To compute $E(0, 2)$, we need the p_0 and p_1 values. Since there is only a single way to get to $(0, 2)$ (two consecutive YESs) the computation is simple: $p_0(0, 2)$ is $(1 - s)$ (V must be 0) times twice e_0, i.e., 0.02 (thus, $p_0(0, 2) = (1 - s) \times e_0^2$). Similarly, $p_1(0, 2) = s \times (1 - e_1)^2 = 0.5 \times 0.9 \times 0.9 = 0.405$. Thus, $E(0, 2) = 0.02/[0.02 + 0.405] = 0.05$. When there are multiple ways to get to a point (e.g., for $(1, 1)$) the p_0, p_1 computation is more complex, and will be discussed in Section 2.4.

The expected error E for this sample strategy is the error weighed by the probability of getting to each termination point. In this case, it turns out that $E = 0.115$. Notice the difference between the overall error E and the individual termination point errors: The error at $(1, 1)$ is quite high, but because it is not very likely that we end up at $(1, 1)$, that error does not contribute as much to the overall E.

2.3 The Problems

Given input parameters s, e_1, and e_0, the search for a "good" or "optimal" strategy can be formulated in a variety of ways. Since we want to ensure that the strategy terminates, we enforce a threshold m on the maximum number of questions we can ask for any item (which is nothing but a maximum budget for any item that we want to filter).

We start with the problem we will focus on in this paper:

PROBLEM 1 (CORE). *Given an error threshold τ and a budget threshold per item m, find a strategy that minimizes C under the constraint $E < \tau$ and $\forall(x, y)\ C(x, y) < m$.*

An alternative would be to constrain the error at each termination point:

PROBLEM 2 (CORE: PER-ITEM ERROR). *Given an error threshold τ and a budget threshold per item m, find a strategy that minimizes C under the constraint $\forall(x, y)\ E(x, y) < \tau$ and $C(x, y) < m$.*

$$p_0(x,y) = \begin{cases} p_0(x-1,y)(1-e_0) + p_0(x,y-1)e_0 & \text{if } \neg\mathsf{Term}(x,y-1) \wedge \neg\mathsf{Term}(x-1,y) \\ p_0(x,y-1)e_0 & \text{if } \neg\mathsf{Term}(x,y-1) \wedge \mathsf{Term}(x-1,y) \\ p_0(x-1,y)(1-e_0) & \text{if } \mathsf{Term}(x,y-1) \wedge \neg\mathsf{Term}(x-1,y) \\ 0 & \text{if } \mathsf{Term}(x,y-1) \wedge \mathsf{Term}(x-1,y) \end{cases}$$

$$p_1(x,y) = \begin{cases} p_1(x,y-1)(1-e_1) + p_1(x-1,y)e_1 & \text{if } \neg\mathsf{Term}(x,y-1) \wedge \neg\mathsf{Term}(x-1,y) \\ p_1(x,y-1)(1-e_1) & \text{if } \neg\mathsf{Term}(x,y-1) \wedge \mathsf{Term}(x-1,y) \\ p_1(x-1,y)e_1 & \text{if } \mathsf{Term}(x,y-1) \wedge \neg\mathsf{Term}(x-1,y) \\ 0 & \text{if } \mathsf{Term}(x,y-1) \wedge \mathsf{Term}(x-1,y) \end{cases}$$

Figure 2: Recursive Equations for p_0 and p_1

In Problem 2 we ensure that the error is never above threshold, even in very unlikely executions. This formulation may be preferred when errors are disastrous, e.g., if our filter is checking for patients with some disease, or for defective automobiles. However, in other cases we may be willing to tolerate uncertainty in some individual decisions (i.e., high $E(x,y)$ at some points), in order to reduce costs, as long as the overall error E is acceptable. For instance, say we are filtering photographs that will be used by an internet shopping site. In some unlikely case, we may get 10 YES and 10 NO votes, which may mean we are not sure if the photo satisfies the filter. In this case we may prefer to stop asking questions to contain costs and make a decision, even though the error rate at this point will be high no matter what we decide. In Section 7 we study the price one pays (number of questions) for adopting the more conservative approach of Problem 2 under the same error threshold.

Instead of minimizing the overall cost, one could minimize the *maximum* cost at any given point, as in the next problem:

PROBLEM 3 (MAXIMUM COST). *Given an error threshold τ, find a strategy that minimizes the maximum value of $C(x,y)$ (over all points), under the constraint $E < \tau$.*

Another variation is to minimize error, given some constraint on the number of questions. We specify two such variants next. In the first variant we have a maximum budget for each item we want to filter, while in the second variant we have an additional constraint on the expected overall cost.

PROBLEM 4 (ERROR). *Given a budget threshold per item m, find a strategy that minimizes E under the constraint $\forall(x,y)\ C(x,y) < m$.*

PROBLEM 5 (ERROR - II). *Given a budget threshold per item m and a cost threshold α, find a strategy that minimizes E under the constraints $\forall(x,y)\ C(x,y) < m$ and $C < \alpha$.*

2.4 Computing Probabilities at Grid Points

In this subsection we show how to compute the p_0 and p_1 values defined in the previous subsection. We focus on a deterministic strategy (probabilistic strategies are discussed in Section 4).

We compute the p values recursively, starting at the origin. In particular, note that $p_0(0,0)$ is $(1-s)$ and $p_1(0,0)$ is s. (For instance, $p_0(0,0)$ is the probability that the item does not satisfy the filter and the strategy visits the point $(0,0)$ — which it has to.) We can then derive the probability p_0 and p_1 for every other point in the grid as shown in Figure 2. (Recall that $\mathsf{Term}(x,y)$ means that (x,y) is a termination point, either Pass or Fail.)

To see how these equations work, let us consider the first case for p_0, where neither $(x-1,y)$ nor $(x,y-1)$ are termination points. Note that we can get to (x,y) in two ways, either from $(x,y-1)$ on getting an extra YES, or from $(x-1,y)$ on getting an extra NO. Thus, the probability of an item not satisfying the filter and getting to (x,y) is the sum of two quantities: (a) the probability of the item not satisfying the filter and getting to $(x,y-1)$ and getting an extra YES, and (b) the probability of the item not satisfying the filter and getting to $(x-1,y)$ and getting an extra NO. The probability of getting an extra YES given that the item does not satisfy the filter is precisely e_0, and the probability of getting a NO is $(1-e_0)$. We can write similar equations for p_1, as shown in the figure.

Given values for p_0 and p_1, we can use the definitions of Section 2.1 to compute the errors and costs at each point, and the overall error and cost. Note that we do not have to compute the p values at all points, but only for the reachable points, as all other points have zero error $E(x,y)$ and cost $C(x,y)$.

3. DETERMINISTIC STRATEGIES

This section develops algorithms that find effective deterministic strategies for Problem 1.

3.1 The Paths Principle

Given a deterministic strategy, using Equations 1, 2, 3 and Figure 2 given earlier, it is easy to see that the following theorem holds:

THEOREM 3.1 (COMPUTATION OF COST AND ERROR). *The expected cost and error of a strategy can be computed in time proportional to the number of reachable grid points.*

Based on the above theorem, we have a brute force algorithm to find the best deterministic strategy, namely by examining strategies corresponding to all possible assignments of Pass, Fail or Continue (i.e., continue asking questions) to each point in $(0,0)$ to (m,m). (There are 3^m such assignments.) Evaluating cost and error for each strategy takes time $O(m^2)$ using the recursive equations. We select the one that satisfies the error threshold, and minimizes cost. We call this algorithm naive3.

Note that some of these strategies are not terminating. However, termination can also be checked easily for each strategy considered in time proportional to the number of reachable grid points in the strategy. (If the p_0 and p_1 values at all points on the line $x+y=m'$, for some $m' \leq m$ is zero, then the strategy is terminating.)

THEOREM 3.2 (BEST STRATEGY: NAIVE3). *The* naive3 *algorithm finds the best strategy for Problem 1 in $O(m^2 3^m)$.*

We are able to reduce significantly the search space of the naive algorithm by excluding the provably suboptimal strategies. Our exclusion criterion is based on the following fundamental theorem.

THEOREM 3.3 (PATHS PRINCIPLE). *Given s, e_1, e_0, for every point (x,y), the function $R(x,y) = p_0(x,y)/(p_0(x,y) + p_1(x,y))$ is a function of (x,y), independent of the particular (deterministic or probabilistic) strategy.*

PROOF. Consider a single sequence of x No answers and y Yes answers. The probability that an item satisfies the filter, and gets the particular sequence of x No answers and y Yes answers is precisely

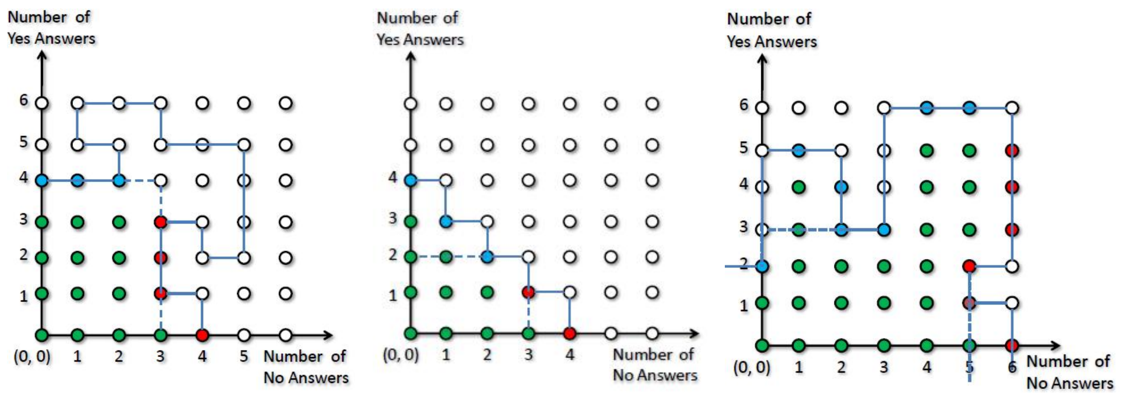


Figure 3: (a) A Shape (b) Triangular Strategy Corresponds to a Shape (c) Ladder Shape Pruning

$a = s \times e_1^x \times (1 - e_1)^y$, while the probability that an item does not satisfy the filter and gets the same sequence is $b = (1 - s) \times e_0^y \times (1 - e_0)^x$. The choice of strategy may change the number of such paths to the point (x, y), however the fraction of p_0 to $p_0 + p_1$ is still $b/(a + b)$. (Note that each path precisely adds a to p_1 and b to p_0. Thus, if there are r paths, $p_0/(p_0 + p_1) = (r \times b)/[r \times (a + b)] = b/(a + b)$.) It can be shown that the proof also generalizes to probabilistic strategies. (In that case, r can be a fractional number of paths.) □

Intuitively, this theorem holds because the strategy only changes the number of paths leading to a point, but the characteristics of the point stay the same.

Using the previous theorem, we have the result that in order to reduce the error, for every termination point (x, y), Passing or Failing is independent of strategy, but is based simply on $R(x, y)$.

THEOREM 3.4 (FILTERING INDEPENDENT OF STRATEGY). *For every optimal strategy, for every point (x, y), if $\mathsf{Term}(x, y)$ holds, then:*

- *If $R(x, y) > 1/2$, then $\mathsf{Fail}(x, y)$*
- *If $R(x, y) < 1/2$, then $\mathsf{Pass}(x, y)$*

PROOF. Given a strategy, let there be a point (x, y) for which $\mathsf{Pass}(x, y)$ holds, but $R(x, y) > 1/2$. Let the error be:

$$E = E_0 + E(x, y)(p_0(x, y) + p_1(x, y))$$

(We split the error into two parts, one dependent on other terminating points, and one just dependent on (x, y).) Currently, since (x, y) is Pass, $E(x, y)$ is $p_0(x, y)/(p_0(x, y) + p_1(x, y))$. Thus, $E = E_0 + p_0(x, y)$. If we change (x, y) to Fail, we get $E' = E_0 + p_1(x, y)$, then $E' < E$. Thus, by flipping (x, y) to Fail, we can only reduce the error. Similarly, if $R(x, y) < 1/2$ and $\mathsf{Fail}(x, y)$ holds, we can only reduce the error by flipping it to $\mathsf{Pass}(x, y)$. □

Thus, we only need to consider 2^m strategies, namely those where for each point, we can either set it to be a continue point or a termination point, and if it is a termination point, then using the previous theorem, we can infer whether it should be Pass or Fail.[1] The algorithm that considers all such 2^m strategies is called naive2. Thus, we have the following theorem:

THEOREM 3.5 (BEST STRATEGY: NAIVE2). *The* naive2 *algorithm finds the best strategy for Problem 1 in $O(m^2 2^m)$.*

[1]Note that there is a subtle point here. When $R(x, y) = 1/2$ then the point can either be a Pass or a Fail point, with equal effect. (It will contribute the same amount of error either way.) For simplicity we assume it to always return Pass if this holds.

3.2 Shapes and Ladders

In practice, considering all 2^m strategies is computationally feasible only for very small m. In this section, we design algorithms that only consider a subset of these 2^m strategies and thereby can only provide an approximate solution to the problem (i.e., the expected cost may be slightly greater than the expected cost of the optimal solution). The subset of strategies that we are interested in are those that correspond to *shapes*. We will describe an efficient way of obtaining the best strategy that corresponds to a shape.

Shapes: A *shape* is defined by a connected sequence of (horizontal or vertical) segments on the grid, beginning at a point on the y-axis, and ending at a point on the x axis, along with a special point somewhere along the sequence of segments, called a *decision point*. We also assume that each segment intersects at most with two other lines, namely the ones preceding and following it in the sequence. As an example, consider the shape in Figure 3(a) (ignore the dashed lines for now) This shape begins at (0, 4), has a sequence of 14 segments, and ends at (4, 0). The decision point (not shown in the figure) is (for example) at (5, 5). As seen in the figure, the segments are allowed to go in any direction (up/down or left/right).

Each shape corresponds to precisely one strategy, namely the one defined as follows:

- For each point in the sequence of segments starting at the point on the y axis, until and including the decision point, we color the point blue (i.e., we designate the point as Pass). In the figure, all points in the sequence of segments starting from (0, 4) until and including (5, 5) are colored blue.
- For each point in the sequence of segments starting at the point after the decision point, until and including the point on the x axis, we color the point red (i.e., we designate the point as Fail). In the figure, all points after (5, 5) on the sequence of segments are colored red.
- For all the points inside or on the shape that are reachable, we color them green; else we color them white. (Some points colored blue or red previously may actually be unreachable and will be colored white in this step.) In the figure, some of the blue points, such as (2, 5), (1, 5), (1, 6), . . ., (5, 5), and some of the red points (5, 4), . . ., (4, 3), and (4, 1) are colored white, while some of the unreachable points inside the shape, such as (3, 4) and (4, 4) are colored white as well. The reachable points inside the shape, like (1, 1) or (2, 3) are colored green.

The strategies that correspond to shapes form a large and diverse class of strategies. In particular, the triangular strategy and rectangular strategy both correspond to shapes. Consider Figure 3(b), which depicts a shape corresponding to the triangular strategy of

Figure 1(a). (Again, ignore dashed lines in the figure.) The shape consists of eight connected segments, beginning at (0, 4) and ending at (4, 0), each alternately going one unit to the right or down. The decision point in this case is the point (3, 2). Note that all points in the segments leading up to (3, 2) are either blue or white (unreachable), while all points in the segments after (3, 2) are red or white. In this case, all the points on the interior of the shape are continue points.

The rectangular strategy of Figure 1(b) corresponds to the shape formed by two segments, one from (0, 4) to (4, 4) and one from (4, 4) to (4, 0). The point (4, 4) is the decision point. As yet another example, consider Figure 3(c). If we consider the shape corresponding to the solid lines, we have a segment from (0, 2) to (0, 5), another from (0, 5) to (2, 5), and so on until (6, 6) — which is the decision point. Then we have five segments from (6, 6) to (0, 6). Once again, notice that some of the points before the decision point in the sequence of segments, such as (3, 4) and (2, 5) are unreachable, and some points after the decision point, like (6, 1) and (6, 2) are unreachable. The decision point is unreachable as well. In this case, all internal points are reachable (and thus colored green).

As an example of a strategy that does not correspond to a shape, if the strategy in Figure 3(c) had an additional terminating point, say Fail at point (1, 1), then it can never correspond to a shape.

Why shapes? One objection one may have to studying shapes is that the best strategy corresponding to shapes may be much worse than the best strategy overall.

However, the properties that shapes obey make intuitive sense. First, note that the strategies that correspond to shapes only have terminating points on the "boundary" of the strategy, and not on the interior. This makes sense because it is not worthwhile to have a termination point inside the boundary of termination points, since we might as well move the boundary earlier. Second, the strategies have a single decision point; this makes sense because it is not useful to alternate between red and blue points on the boundary, since the more YES answers we get relative to NO answers, the more likely the item should satisfy the filter, hence we should be able to improve the strategy by converting it to one with a single point where the colors change.

In addition, we found that over 100 iterations of the naive2 algorithm for random instances of the parameters s, e_1, e_0 and m, by inspection all of the optimal strategies corresponded to shapes. In addition, as we will see in the experiments, on varying parameters, the best strategy given by naive2 is no better than the best strategy corresponding to a shape.

Hence, we pose the following conjecture:

CONJECTURE 3.6. *Given a problem, the best deterministic strategy is one that corresponds to a shape.*

Proving this conjecture remains open.

Ladder Shapes: From all strategies that correspond to shapes, if we wanted to find the best strategy, we can prove that we only need consider the subset of shapes that we call *ladder shapes*. A ladder shape is formed out of two *ladders* connected at the decision point. We first define a *ladder* to be a connected sequence of (flat or vertical) segments connecting grid points, such that the flat lines go "right", i.e., from a smaller x value to a larger x value, and the vertical lines go "up", i.e., from a smaller y value to a larger y value. As an example, in Figure 3(c), the sequence of (dashed and solid) segments (0, 2)-(0, 3)-(3, 3)-(3, 6)-(6, 6) forms a ladder, while the sequence of segments (0, 2)-(0, 5)-(2, 5)-(2, 3)-(3, 3)-(3, 6)-(6, 6) is not a ladder because (2, 5) to (2, 3) goes from a larger y value to a smaller y value. As another example, the sequence of solid segments from (0, 4) to (4, 0) in Figure 3(b) is not a ladder because the vertical segments go from a larger y value to a smaller y value, while the segment (0-2)-(3,2) is a ladder.

We define the set of ladder shapes to be those shapes that contain a decision point and two ladders, i.e., the connected sequence of segments from the point on the y axis to the decision point forms one ladder and the connected sequence of segments from the point on the x axis to the decision point forms the second ladder. Thus, ladder shapes are a subset of the set of all shapes. Intuitively, ladder shapes are the shapes that we would expect to be optimal: the shape is smaller at the sides (close to the x and y axis, where we are more certain whether the item satisfies the filter or not), and larger close to the center (away from both the x and y axis, where we are more uncertain about the item).

As before, the strategy that corresponds to a ladder shape is the one formed by coloring all the points in the "upper" ladder blue, and all the points in the "lower" ladder red, then coloring all remaining reachable points inside the shape green, and coloring all unreachable points inside or on the boundary of the shape white. In the following, we provide a few examples of how we can convert any shape into a ladder shape such that the strategy corresponding to the ladder shape has the same or lower cost than the strategy corresponding to the shape. Since the set of ladder shapes is a subset of all possible shapes, if we want to find the best strategy corresponding to a shape, we can thus focus on ladder shapes instead of all shapes. Thus, we have the following theorems:

THEOREM 3.7 (TRANSFORMATION). *Any shape can be converted into a ladder shape yielding lesser cost and the same error.*

We now describe some examples of how the conversion algorithm works. The algorithm along with an informal proof can be found in the appendix. The algorithm essentially prunes redundant portions of the shape to give a ladder shape.

THEOREM 3.8 (BEST SHAPE). *For problem 1, the best strategy from the set of shapes has equal cost to the best strategy from the set of ladder shapes.*

The above theorem gives us an algorithm, denoted ladder, which considers a small subset of the set of all shapes, namely all the ladder shapes. Note that this algorithm is still worst-case exponential; however, as we will see in the experiments, this algorithm performs reasonably well in practice, and in particular, much better than naive2.

Examples of Converting Shapes to Ladder Shapes: First, consider the triangular strategy shown in Figure 3(b). As it stands, the shape (formed from the solid lines in the figure) is not a ladder shape, since the sequence of segments leading to the decision point (3, 2) from the point on the x axis as well as the point on the y axis don't form ladders. While the ladder from the y axis has segments that go "down" instead of "up", the ladder from the x axis has segments that go "left" instead of "right". In the strategy corresponding to the shape, note that asking questions at points above the line $y = 2$ is redundant, because once we cross $y = 2$ (i.e., 2 Yes answers), we will always reach a Pass point. Similarly, notice that asking questions at points on the right of the line $x = 3$ is redundant. Thus, we can convert this triangular strategy into a rectangular strategy with the same error and lower cost simply by pruning the regions to the top and to the right of the decision point, and having termination points earlier. Notice that this corresponds to the ladder shape formed by the two dashed ladders (one from (0, 2) to (3, 2) and one from (3, 0) to (3, 2), with the decision point (3, 2)). Thus, the shape (giving the triangular strategy) can be converted into a ladder shape (giving a rectangular strategy) with lower cost and the same error.

As another example, consider Figure 3(c). Here the solid blue line represents a shape corresponding to the strategy formed by the blue, green and red dots. The shape has 10 lines: (0, 2)-(0, 5)-(2, 5)-(2, 3)-(3, 3)-(3, 6)-(6, 6) (which is also the decision point), (6, 6)-(6, 2) and so on. Now consider the shape corresponding to the dashed blue line in the figure. (This shape is the same as the solid shape, except for the portion (0, 3) to (3, 3) which bypasses the segment portions (0, 3)-(0, 5)-(2, 5)-(2, 3)-(3, 3), and the portion (5, 0) to (5, 1) which bypasses the segment portions (6, 0)-(6, 1)-(5, 1)). Notice that this shape corresponds to a ladder shape (with one ladder beginning at (0, 2) and ending at (6, 6), and another beginning at (5, 0) and ending at (6, 6), with a decision point at (6, 6)). This ladder shape corresponds to the strategy where there is a blue point at (1, 3) and a red point at (5, 0). Notice that for the strategy that corresponds to the shape, asking questions at (1, 3) and (1, 4) is redundant because the item will "Pass" no matter what answers we get at (1, 3) and (1, 4). Thus, moving the segment portion down to (1, 3) and to (5, 0) gives us a ladder shape that corresponds to a strategy that asks fewer questions to obtain the same result.

As yet another example, consider Figure 3(a), in this case, the shape corresponding to the solid lines in the figure (with decision point (5, 5)), can be replaced by the ladder shape corresponding to the two ladders (0, 4)-(3, 4) and (3, 0)-(3, 4), with decision point (3, 4).

Thus, we have shown some examples of how we can convert shapes into ladder shapes such that the strategy corresponding to the ladder shape has lower cost than the strategy corresponding to the shape.

4. PROBABILISTIC STRATEGIES

In this section, we consider probabilistic strategies. Recall that a probabilistic strategy is again represented in a grid, however, each point has a triple (r_1, r_2, r_3) corresponding to the probability of returning Pass (blue), Fail (red), or Continue asking questions (green).

Since the Paths Principle (Theorem 3.3) also holds for probabilistic strategies, at least one of r_1 or r_2 must be 0 at each point (x, y). We let $a(x, y) = r_3$ be the probability that we continue to ask questions at (x, y), and $b(x, y)$ be $1 - a(x, y) = r_1 + r_2$. (This is the probability that the strategy terminates at that point.)

We can pose Problem 1 as a set of constraints, where the objective is to minimize the expected cost C, given a constraint that $E < \tau$, along with some additional constraints, which are essentially the counterparts of the equations described in Section 2.

- We have the probabilistic counterpart of Equation 1:

$$\begin{aligned}&\forall (x,y); x+y \le m:\\ &E(x,y) = b(x,y) \times \min(R(x,y), 1 - R(x,y))\end{aligned} \tag{4}$$

  (Recall that $R(x, y)$ is defined in Theorem 3.3.) Note that the error at a certain point is simply the probability that the strategy terminates at that point, times the smaller of the two error probabilities $R(x, y)$ and $1 - R(x, y)$ (since the Paths Principle tells us that we would always choose whichever of Pass or Fail has lower error probability). Note that $R(x, y)$ is a constant, independent of the strategy.

- The cost at a given point is simply the probability that the strategy terminates at that point times the total number of questions asked to get to the point.

$$\forall (x,y); x+y \le m : C(x,y) = b(x,y) \times (x+y) \tag{5}$$

- The error and cost equations stay the same as equations 2 and 3.

$$E = \sum_{(x,y);x+y\le m} E(x,y) \times [p_0(x,y) + p_1(x,y)] \tag{6}$$

$$C = \sum_{(x,y);x+y\le m} C(x,y) \times [p_0(x,y) + p_1(x,y)] \tag{7}$$

- The counterpart of the equations in Figure 2 is simpler since we ask an additional question at $(x, y - 1)$ with probability $a(x, y - 1)$ and at $(x - 1, y)$ with probability $a(x - 1, y)$.

$$\begin{aligned}\forall (x,y);\quad & x+y \le m:\\ p_0(x,y)\quad & = e_0 \cdot p_0(x, y-1) \cdot a(x, y-1)\\ & +(1-e_0)\, p_0(x-1,y) \cdot a(x-1,y)\end{aligned} \tag{8}$$

$$\begin{aligned}\forall (x,y);\quad & x+y \le m:\\ p_1(x,y)\quad & = e_1 \cdot p_1(x-1, y) \cdot a(x-1, y)\\ & +(1-e_1)\, p_1(x,y-1) \cdot a(x,y-1)\end{aligned} \tag{9}$$

- In addition, we have the following constraints:

$$\forall (x,y); x+y = m: \ \ b(x,y) = 1 \tag{10}$$

  The constraint above forces the strategy to terminate at m questions.

$$\forall (x,y); x+y \le m: \ \ a(x,y) + b(x,y) = 1 \tag{11}$$

  This constraint simply forces the probabilities of termination and continuation at each point on the grid to add up to one.

This program is not linear, due to constraints 6, 7, 8 and 9 (all of which involve a product of two variables). A key technical result of our work is that we can use the Paths Principle (Theorem 3.3) to transform the program into a linear program.

Transformed Program: We introduce a new pair of variables $cPath$ and $tPath$ to replace p_0, p_1, a, b for every point in the grid. The variable $cPath(x, y)$ corresponds to the (fractional) number of paths in the strategy from $(0, 0)$ to (x, y) that continue onwards beyond (x, y), while $tPath(x, y)$ is the number of paths in the strategy that terminate at (x, y). Thus, $cPath(x, y) + tPath(x, y)$ represents the number of paths reaching (x, y). For instance $cPath(x, y) = 0$ implies that all paths reaching (x, y) terminate at (x, y).

We let $S_1(x, y) = s \times e_1^x \times (1-e_1)^y$ (i.e., the probability that the item satisfies the filter, and we get a given sequence of x no answers and y yes answers), and $S_0(x, y) = (1 - s) \times e_0^y \times (1 - e_0)^x$ (i.e., the probability that the item does not satisfy the filter, and we get a given sequence of x no answers and y yes answers). Note that $R(x, y) = S_0(x, y)/(S_0(x, y) + S_1(x, y))$. Note also that S_0 and S_1 are constants.

The following relationships are immediate:

$$p_0(x,y) = S_0(x,y) \times (cPath(x,y) + tPath(x,y))$$

$$p_1(x,y) = S_1(x,y) \times (cPath(x,y) + tPath(x,y))$$

These relationships hold because the probability of getting to a point when the item satisfies the filter (or not) is simply the total number of paths times the probability of a single path when the item satisfies the filter (or not).

Additionally, since the probability a is simply the fraction of paths that continue beyond (x, y), we have:

$$a(x,y) = cPath(x,y)/(cPath(x,y) + tPath(x,y))$$

$$b(x,y) = tPath(x,y)/(cPath(x,y) + tPath(x,y))$$

Now we describe how to rewrite the equations in terms of $cPath$ and $tPath$.

- Constraints 4, 5, 6 and 7 transform into:

$$E = \sum_{(x,y);x+y\leq m} tPath(x,y) \times \min(S_0(x,y), S_1(x,y))$$

$$C = \sum_{(x,y);x+y\leq m} tPath(x,y)(x+y)(S_0(x,y) + S_1(x,y))$$

(Recall that S_0, S_1 are constants, so the constraints are linear.) In other words, E is precisely the number of paths leading to the point that terminate at that point, times the smaller of the two error probabilities. The cost C is simply the cost for all paths terminating at (x,y) (each such path has probability $S_0 + S_1$).

- Constraints 8 and 9 transform into:

$$\forall(x,y); x+y \leq m : cPath(x,y) + tPath(x,y) = cPath(x,y-1) + cPath(x-1,y)$$

In other words, the number of paths into (x,y) are precisely those that come from $(x,y-1)$ and $(x-1,y)$

- We replace constraint 10 with the following, which implies that no paths go beyond $x+y=m$.

$$\forall(x,y); x+y=m : \ cPath(x,y) = 0$$

- We also have the constraint that there is a single path to $(0,0)$, i.e.

$$cPath(0,0) + tPath(0,0) = 1$$

No additional constraints exist.

Since linear programs can be solved in time quadratic in the number of variables and constraints, we have the following theorem:

THEOREM 4.1 (BEST PROBABILISTIC STRATEGY). *The best probabilistic strategy for Problem 1 can be found in $O(m^4)$.*

We denote the algorithm corresponding to the linear program above as linear.

5. OTHER FORMULATIONS

5.1 Problem 2

For Problem 2, we can show that we simply need to compute $R(x,y)$ for every point in $(0,0)$ to (m,m), bottom up, and for every point where we find that $\min(R(x,y), 1-R(x,y)) < \tau$, we make the point a terminating point, returning Fail if $R(x,y) \leq 0.5$ and Fail otherwise.

In fact, we can actually terminate earlier if we find that p_0 and p_1 are 0 for all points $(x,y) : x+y = m'$ at some $m' < m$. In this case, we do not need to proceed beyond points along $x+y=m'$.

Note that a feasible strategy that terminates and satisfies $E(x,y) < \tau$ for every terminating point may not exist.

Thus, we have the following theorem:

THEOREM 5.1. *The best strategy for Problem 2 can be found in $O(m^2)$.*

5.2 Problems 3 and 4

We begin by considering Problem 4 first. This problem formulation is identical to Maximum A-Posteriori (MAP) estimation. In this case we simply ask all m questions (since there is no gain to asking fewer than m questions). We can estimate the p_0 and p_1 at all points along $x+y=m$ for this triangular strategy. If $p_0 > p_1$, we return Fail at that point and Pass otherwise. We can then estimate the error E over all the terminating points.

Thus, we have the following theorem:

THEOREM 5.2. *The best strategy for Problem 4 can be found in $O(m^2)$.*

(We can actually compute the best strategy in $O(m)$ if binomial expressions involving m can be computed in $O(1)$ time.)

Subsequently, we can solve Problem 3 by performing repeated doubling of the maximum cost m, until we find a triangular shape for which $E < \tau$, followed by binary search between the two thresholds m and $m/2$. Thus, we have the following theorem:

THEOREM 5.3. *The best strategy for Problem 4 can be found in $O(m^2 \log m)$.*

(We can actually compute the best strategy in $O(m \log m)$ if binomial expressions involving m can be computed in $O(1)$ time.)

5.3 Problem 5

For Problem 5, we can use the same linear programming formulation as in Section 2.4, except that we constrain C and minimize E. We thus have the following:

THEOREM 5.4. *The best probabilistic strategy for Problem 5 can be found in $O(m^4)$.*

6. MULTIPLE FILTERS

Now, we consider the case when we have independent filters $f_1, f_2, \ldots, f_k$, with independent selectivities $s_1, s_2, \ldots, s_k$, and independent error rates e_{10}, e_{11}, e_{20}, e_{21}, …, e_{k0}, e_{k1}, where e_{i0} is the probability that a human answers YES when the item actually does not pass the ith filter, and e_{i1} is the probability that the human answers NO when the item actually does pass the ith filter. Note that the multiple filters problem is much harder than the single filter one, since a strategy not only must decide when to continue asking questions, but must also now decide *which* question to ask next (i.e., for what filter). The problem becomes even harder when the filters are correlated, however, we leave it for future work.

We can visualize the multiple filters case as a $2k$ dimensional grid, where each point $(x_{10}, x_{11}, x_{20}, x_{21}, \ldots, x_{k0}, x_{k1})$ corresponds to the number of NO and YES answers received from humans for each of the filters. (x_{i0} indicates the number of NO answers for the ith filter, and x_{i1} indicates the number of YES answers for the ith filter.)

In its most general form, the multiple filters problem allows us, at any point on this $2k$-dimensional grid, to ask a question corresponding to any of the k filters. Thus, each point can correspond to "Pass", "Fail", or "Ask ith Filter", where i can be one from $1 \ldots k$. Using a general form of the Paths Principle (Theorem 3.3), if a point is a terminating point, we can infer whether it should be a Pass or a Fail point.

We can in fact represent the problem as a linear program, generalizing the linear program in Section 4. The counterpart of the variables $tPath$ and $cPath$ are $tPath$ and $cPath_1, \ldots, cPath_k$, representing respectively, the number of paths terminating at a given point and the number of paths continuing in the direction of each of the k filters. Similarly, the total number of paths coming into a point is simply the paths coming in from each of the k directions.

THEOREM 6.1. *The best probabilistic strategy for the multiple filters version of Problem 5 can be found in $O(m^{4k})$.*

This algorithm is exponential in the number of filters k, which may not be very large. However, any algorithm whose output is a strategy using our visualization would need $\Omega(m^{2k})$, since we need to provide a decision for each point in the $2k$ dimensional cube of size m^{2k}. It remains to be seen if we can find an optimal or approximately optimal strategy whose representation is smaller.

7. EXPERIMENTS

The goal of our experiments is to study the runtime and expected cost and errors of our algorithms with respect to other naive and approximate algorithms. We continue to focus on Problem 1.

In our experiments we explored wide ranges of values for our parameters m, e_0, e_1, τ, s. In some cases we manually selected the values, to study scenarios that interested us or to study extreme cases in the parameter space. In other cases, we synthetically generated random instances of the parameter values (over given ranges), to explore the average behavior. Since we do not have space to report all of our results, we only report some representative ones, and we explain how they support the findings that we state in boxes below.

Algorithms:

We experimented with the following exact deterministic algorithms:

- naive3: The naive algorithm that considers all 3^m strategies.
- naive2: The naive algorithm that considers all 2^m strategies (after pruning strategies that violate the Paths Principle).

We also experimented with the following heuristic deterministic algorithms:

- ladder: This algorithm returns the best strategy corresponding to a ladder shape. This algorithm always returns a better deterministic strategy than the heuristic proposed in [9], as we will see in Section 8.
- growth: This greedy algorithm "grows" a strategy until the constraints are met. It begins with the null strategy at $(0, 0)$ (i.e., terminate and return Pass or Fail). Then, the algorithm "pushes the boundary ahead". In other words, in each iteration, the algorithm in turn considers moving each termination point (x, y) to $(x + 1, y)$ and $(x, y + 1)$ and computes the ratio of change in cost to change in error. The algorithm decides to move the termination point that yields the smallest increase in this ratio. This "pushing" continues until the error constraint is satisfied.
- shrink: This greedy algorithm "shrinks" a strategy until the cost cannot be decreased any longer. It begins with the triangular strategy that asks all m questions, and "pushes the boundary in". In other words, in each iteration, the algorithm for each terminating point (x, y) in turn, considers adding a terminating point at $(x, y-1)$ or $(x-1, y)$ and computes the ratio of the change in cost to change in error. The algorithm decides to add a terminating point that yields the largest increase in this ratio. The "shrinking" continues as long as the error constraint is satisfied.
- rect: This algorithm tries all rectangular strategies that fit in (m, m).

In addition, we have the optimal probabilistic algorithm:

- linear: This algorithm returns the strategy computed by the linear program in Section 4.

Also, we compared our algorithms against the best algorithm for Problem 2.

- point: This algorithm ensures that at every termination point, $E(x, y) < \tau$. Thus, this algorithm is optimal for Problem 2. When the algorithm cannot find a feasible solution for Problem 2, we modify the solution to ensure that at least the cost constraint $C(x, y) < m$ is satisfied. That is, for an infeasible solution, we add termination points along the boundary $x + y = m$, if those points are reachable.

Note that there may be parameters for which some of the algorithms return an infeasible solution (i.e., where the error constraint is violated). It can be shown that for all algorithms except growth and point, either all algorithms return feasible solutions or none of them do. Algorithms growth and point, in addition to failing whenever other algorithms fail, also fail in some other cases.

Comparison of Heuristic Deterministic Algorithms:

Results on Varying m: ladder *results in large cost savings compared to other heuristic deterministic algorithms, and furthermore its cost decreases as m increases*

Figure 4(a) presents the results of an experiment that supports this finding. In this scenario, the parameters are $s = 0.6, e_0 = 0.2, e_1 = 0.25, \tau = 0.05$, and m (horizontal axis) is varied from 8 to 16. The vertical axis shows the expected cost C returned by the strategy found by the heuristic deterministic algorithms. For instance, when m is 14, the cost for ladder is about 3.85 (i.e., we need around 3.85 questions on average to get the desired expected error), growth is about 3.9 and shrink is about 4. The plot for rect is not shown, but rect is a straight line at about 5.6.

Note that even these small differences in expected cost can result in major cost savings overall. If there are a million items that need to be filtered, where each question costs 10 cents and m is 14, the ladder algorithm results in at least $0.05 * 10^6 * 0.1 = \$5000$ of savings over the shrink and growth algorithm, and a whopping $100000 of savings over the rect algorithm.

While at first one might think that increasing the question limit m will *increase* the overall cost, observe that in reality the opposite is true for the ladder and rect algorithms. The reason is that as m increases, the number of shapes available for consideration strictly increases, giving the optimizer more choices. The same cannot be said of the shrink and growth algorithms, since these are both heuristic greedy search algorithms that can get stuck in local minima. For instance, the cost for shrink increases as m goes from 10 to 11, and once again from 14 to 15 in the experiment above. Note also that none of the algorithms give a feasible solution when $m < 8$ for the set of constraints.

As mentioned earlier, space constraints prevent us from including multiple results per finding. The extensive additional experiments we performed support all of our findings.

Results on Varying s: growth *sometimes gets stuck in local minima; if not,* shrink *and* growth *outperform* rect.

Our second experiment, depicted in Figure 4(b), illustrates this finding. We fixed parameters $e_0 = 0.2, e_1 = 0.25, \tau = 0.05, m = 10$, and varied s from 0.2 to 0.8 and compared the same algorithms as before. The expected cost is plotted as a function of s. For instance, when s is 0.5, the cost for ladder is about 4.15, growth is about 4.28, rect (not depicted in figure) is about 5.68 and shrink is about 4.35. Once again, ladder performs the best. As expected, the cost increases when s is close to 0.5 since that situation is the most uncertain (and therefore we need to ask more questions). Additionally, when $s < 0.3$ or $s > 0.7$, growth gives an infeasible answer (i.e., it gives a strategy that does not satisfy the constraint on error) — depicted in the graph as the cost being set to ∞. Since the growth strategy does a local search around the origin, it can get stuck in an infeasible local minimum where the error can no longer be reduced by growing the strategy. The algorithm rect is much worse than the other algorithms with an additional expected cost of at least 1 over the other algorithms. However, it does give a feasible solution when growth does not.

Results on Varying e_1: Cost increases superlinearly as e_1 increases for all algorithms.

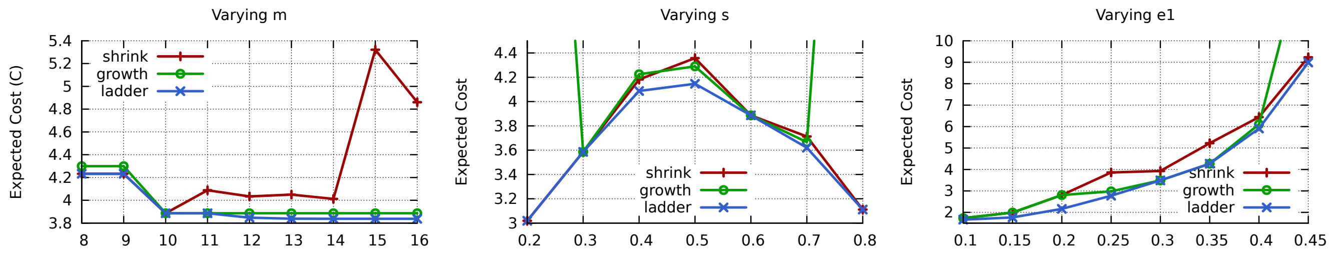


Figure 4: For fixed values of parameters: (a) Varying m (b) Varying s (c) Varying e_1

In the third experiment, depicted in Figure 4(c), we fixed parameters $e_0 = 0.25, s = 0.7, \tau = 0.1, m = 15$, and varied e_1 from 0.1 to 0.45 and compared the algorithms. The expected cost is plotted as a function of e_1. Once again, the ladder algorithm performs the best, while the growth and shrink algorithms each perform well in some situations. Also, the expected cost increases as e_1 increases, in a superlinear fashion. growth once again returns an infeasible answer for large e_1, and there are some points where shrink does extremely poorly compared to ladder, such as $e_0 = 0.25$ (with a difference in expected cost of 1, translating to $> 33\%$ in savings). The experimental results on varying e_0 are similar and therefore not shown.

Comparison of ladder, point, and linear algorithms:

Having shown that ladder is the best heuristic deterministic algorithm (at least in terms of the quality of the results; run time performance is explored later on), the next natural question is how it compares to our other choices.

Results on Varying m: linear *performs even better than* ladder*, both of which perform significantly better than* point*, which sometimes returns infeasible solutions.*

To illustrate our finding, we repeated the same set-up of the experiment in Figure 4(a) — the results are depicted in Figure 5(a). We find that the point algorithm returns an infeasible solution even for the case when $m = 8$ or 9, but for $m = 10$ onwards it returns a feasible solution. This is probably because the point algorithm terminates too early (and has too "narrow" a strategy) when m is small, while the linear and ladder algorithms have a "wider" shape. Interestingly, the cost of the point algorithm also increases as m increases; this is because the strategy looks identical for any m and $m + 1$, except that the termination points along $x + y = m$ may have been moved up to $x + y = m + 1$.

While the point algorithm is infeasible for $m = 9$, linear has a cost of nearly 0.1 less than ladder, representing significant savings in overall cost. In addition, the cost returned by linear only decreases as m increases since a larger space of solutions are considered.

The cost difference between point and linear is greater than what we have observed in our earlier graphs. This difference highlights the benefits of using an expected error bound (Problem 1) as opposed to a point error bound (Problem 2). And as mentioned earlier, if we are filtering one million items, the savings are multiplicative. Thus, for crowdsourcing applications where expected error is adequate, linear (or ladder if a deterministic strategy is desired) is clearly the way to go.

Comparison of Algorithms on Average:

So far we have illustrated our findings with particular scenarios, i.e., particular parameter settings. To further validate our findings, in this subsection we explore the average behavior of our algorithms as we vary parameters randomly over a range of values.

Results on Varying m: (1) The linear *and* ladder *algorithms continue to outperform the other algorithms in our average scenario, especially for large m. (2) For smaller m,* shrink *does quite well.*

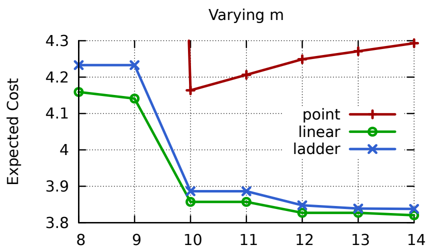
In the first experiment, depicted in Figure 5(b), we compared the algorithms on varying m for 100 synthetic instances where s, e_0, e_1, τ were sampled from $[0, 1]$, $[0.1, 0.4]$, $[0.1, 0.4]$, $[0.005, 0.1]$ respectively. (We made sure that all three algorithms were feasible for each of the random instances used.) The expected cost is then averaged over all 100 instances, and plotted as a function of m. We focus on the best algorithms so far, ladder and linear. We also show shrink for comparison. (rect is always much worse, and growth and point have cases where they returns infeasible answers, and are typically not much better than shrink when they are feasible).

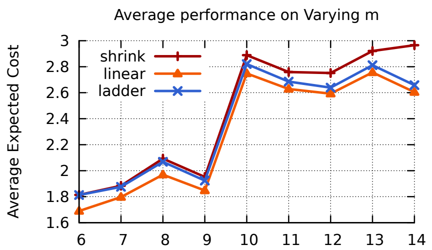
We can see in Figure 5(b) that linear performs much better than all algorithms for all m, with at least a difference of 0.1 on average. For lower m values, for these settings we do not see much of an improvement on average between ladder and shrink, which indicates that we may be able to use shrink for cases when m is small. However, for larger m, we find that the difference between ladder and shrink is at least 0.1. This is because there are more opportunities for optimization for ladder and linear compared to shrink as m increases.

Interestingly, we find that the cost does not strictly decrease for linear and ladder as m increases; this is an artifact of our experimental setup. For each m, we restrict our instances to those for which all algorithms are feasible. Thus, for smaller m we are not considering many of the "harder" cases, which are handled by larger m, instead, we are only consider the "easier" cases, whose expected cost is smaller. On the other hand, allowing a larger m means that we can explore a larger space of solutions. Thus, the variation with m is not as predictable as in the earlier case, except that it seems to gradually increase with some small variations.

Results on Varying m: linear *yields strictly better (lower cost) strategies than* ladder *in a substantial majority of the scenarios. Furthermore,* ladder *outperforms the rest of the deterministic algorithms in a substantial number of scenarios.*

The previous experiment indicates that, on average, we get significant improvements in cost by using linear and ladder. However, we would like to verify if this average scenario is because of a few instances where there is a high difference in cost (while the rest of the instances return the same cost). To see how often we get strictly better strategies by using either ladder or linear, we counted the number of random instances where linear gave a strictly better strategy than the ladder, and the number of cases where ladder gave a strictly better strategy than the rest of the algorithms for the





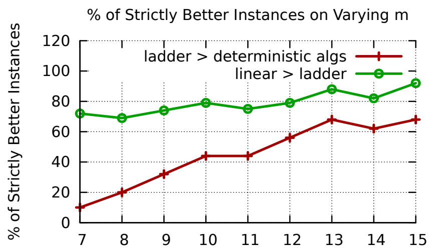
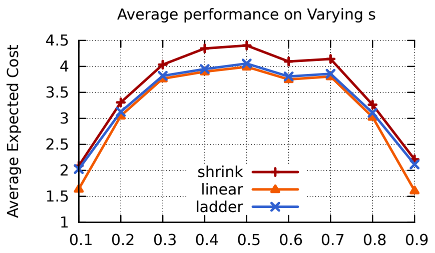
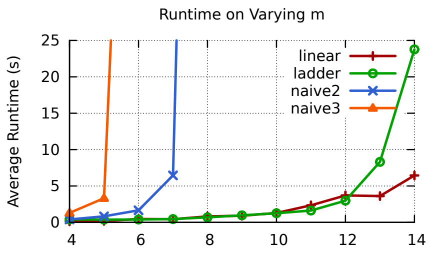


Figure 5: (a) Varying m for fixed values of parameters (b) Varying m on Average (c) Fraction of strictly better shapes





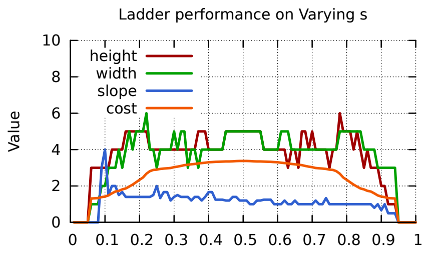


Figure 6: (a) Varying s on average (b) Runtime variation with m (c) Ladder Variation with s

same experimental setup described above. The results are depicted in Figure 5(c).

We find that for $m = 10$, linear gives a strictly better strategy than ladder in 80% of the cases (and in the remaining 20% of the cases, gives a strategy of the same cost as ladder). On the other hand, ladder gives a strictly better strategy than the rest of the algorithms for over 45% of the cases (and in the remaining 55%, gives a strategy of the same cost as the best heuristic algorithm).

Further, the number of scenarios where linear outperforms ladder (and where ladder outperforms the others) continues to increase as m increases. For $m = 15$, linear gives a strictly better strategy than ladder in 90% of the cases, while ladder gives a strictly better strategy than the rest of the algorithms in 65% of the cases.

Results on Varying s: The linear *algorithm outperforms the* ladder *algorithm much more when s is away from* 0.5*, while the* ladder *algorithm outperforms the other algorithms more when s is close to* 0.5*.*

In the second experiment, depicted in Figure 6(a), we compared the algorithms on varying s from 0.1 to 0.9 for 100 synthetic instances where $m = 14$ and e_0, e_1, τ were sampled from $[0.1, 0.4]$, $[0.1, 0.4]$, $[0.005, 0.1]$ respectively. The average expected cost was plotted as a function of s. In this case, we find that linear on average yields a smaller cost than the other two algorithms by around 0.1, but more so when s is away from 0.5. On the other hand, shrink on average yields a larger cost than ladder, and more so when s is closer to 0.5. In fact, when $s = 0.5$ shrink asks 0.5 higher questions on average.

Comparison to Naive Algorithms:

Cost for Random Instances: ladder *performs identically to* naive2 *and* naive3 *for each random instance generated.*

In Section 3.2, we informally argued that any optimal deterministic strategy should be found by the ladder algorithm, although we do not have a formal proof yet. In our next experiment we checked for cost differences between strategies found by ladder and those found by naive2, an exhaustive algorithm that does find the optimal deterministic strategy. (naive3 gives the same result as naive2.) We generated 100 synthetic instances for each m from 4 to 8 (with s, τ, e_1, e_0 all uniformly sampled from $[0, 1]$, $[0.1, 0.4]$, $[0.1, 0.4]$, $[0.005, 0.1]$ respectively). (Beyond 8, naive2 was impractical to use.) We found that the ladder algorithm returns a strategy of the same cost as naive2 for each instance. In fact, in none of the many other experiments we performed did we find an instance where ladder does not return the optimal deterministic strategy. This result is a strong indicator that the ladder algorithm is indeed optimal, however, the proof is still open.

Runtime Comparisons

Runtime results on Varying m: naive3 *and* naive2 *become impractical to use even for very small m, while* ladder *and* linear *are efficient alternatives for m as large as* 15*. For very large $m \approx 40$,* shrink, growth, rect *are able to return a strategy fairly quickly.*

In the first experiment, we compared the average runtimes (in seconds) of the naive algorithms with the best deterministic and best probabilistic algorithms across 100 synthetic instances for each m from 4 to 14 (with s, τ, e_1, e_0 all uniformly sampled from $[0, 1]$, $[0.1, 0.4]$, $[0.1, 0.4]$, $[0.005, 0.1]$ respectively). Comparing average runtimes lets us see how much time we might take for an algorithm for any set of parameters on average. Figure 6(b) shows the average runtimes as a function of m. While naive2 and naive3 take more than 5 minutes even for m as small as 5 and 7 respectively, the ladder algorithm takes less than a minute even until $m = 14$, while the linear algorithm takes even less time. Thus, the optimizations of Section 3 let us design strategies for larger m. (Our implementation of linear uses exact arithmetic, and as a result takes longer. If we were willing to get a slightly worse solution, we may be able to reduce the running time even further.) Note that if we are willing to use a slightly worse deterministic algorithm, we can get strategies for substantially larger values of m. For instance, growth is able to return a solution for $m = 40$ within seconds, while shrink is able to return a solution for $m = 40$ within a minute.

Variation of Ladder Shape: In our final set of experiments we studied two interesting properties of the ladder shapes selected by ladder. We define the *height* and *width* of a ladder shape to be, respectively, the largest distance between the two ladders along the y axis and along the x axis respectively. We also define the *slope* of a ladder shape to be the ratio of the y coordinate to the x coordinate of the decision point. Since there is no counterpart to ladders and decision points in the strategies output by linear, we cannot study them in this way.

Results on Varying s: The optimal ladder shape for ladder *has a decision point closer to the y axis (larger slope) when s is small and closer to the x axis (smaller slope) when s is large. In addition, the closer s is to 0.5, the further away the decision point of the ladder shape is from the origin.*

We fix $e_1 = 0.2, e_0 = 0.2, \tau = 0.05, m = 12$, and vary s in increments of 0.005 from 0 until 1. The results are depicted in Figure 6(c). We find that the slope of the optimal ladder shape moves from 4 all the way down to 0 gradually as we increase s. (Note that we set slope = 0 when the strategy is simply a terminating point at the origin.) This result can be explained as follows: When s is very small, it is unlikely that we will get any YES answers, so more questions need to be asked to verify that an item actually passes the filter, but not when the item does not pass the filter. On the other hand, when s is large, it is unlikely that we will get any NO answers, so more questions are needed to ascertain whether an item fails the filter. Note that cost increases until 0.5 and then decreases, as expected. Height and width are mirror images of each other across $x = 0.5$: this is expected since $e_0 = e_1$, thus the best shape for s is the best shape for $1 - s$.

8. RELATED WORK

The work related to our paper falls under four categories: crowdsourced schema matching, active learning, filtering applications, and statistical hypothesis testing.

Crowdsourced schema matching: Recent work by McCann et. al. [9] has considered the problem of using crowdsourcing for schema matching. The core of the problem is similar: the crowd provides noisy answers, and the goal is to determine whether a match is true or not. The strategy used is the following: ask at least v_1 questions, and stop either when the difference between the number of YES and NO answers reaches a certain threshold δ, or when the total number of questions asked is v_2. The authors prove probabilistic bounds on the maximum error from this strategy, under specific assumptions for the performance of human workers. The proposed strategy can be mapped to a shape in our framework. In fact, we can convert the shape into a ladder shape (as described in Section 3.2), and obtain a new strategy that asks fewer questions while providing the same error guarantees. Moreover, since [9] does not optimize for the number of questions, our cost-optimized algorithms can provide even further improvements.

Active Learning: The field of active learning [16] addresses the problem of actively selecting training data to ask an "oracle" (for instance, the oracle could be an expert user) that would help train a classifier with the least error. Our metric for error (i.e., expected error) is the same as the 0-1 loss used in machine learning.

Typical papers studying active learning do not assume that the oracle makes mistakes, and in any case, repeating the same question to the oracle would typically not help. However, there are some papers that do consider the case when many interchangeable noisy humans can be used as oracles, and we discuss them next.

Sheng et. al. [17] consider the problem of obtaining labels for training data in the context of machine learning. Specifically, the problem is whether to ask for another label for an existing data item or whether to acquire more data items, in order to maximize the utility of the training data set for the machine learning algorithm. Similar work has considered a pool of workers where the most "informed" worker is asked for a label on the fly on the most uncertain item [10, 11], and the accuracy of the worker is learned as labels are obtained. Other work [18, 6] considers a setting where the worker's labels are provided beforehand, and the goal is to infer the labels of items and the accuracy of different workers.

None of the previous studies deal with the problem of optimizing the number of questions asked to the crowd. In large-scale human computation, especially in marketplaces such as Mechanical Turk, this metric is the most critical cost factor that needs to be optimized. In our work so far we have not taken into account knowledge of or discrepancies in worker accuracies, assuming a rapidly changing and replaceable worker pool (as in Mechanical Turk). As future work we will explore incorporating worker accuracy into our theory and algorithms.

Filtering Applications: Several practical applications have used heuristic strategies for filtering, typically a majority vote over a fixed number of workers, in the context of sentiment analysis and NLP tasks [15], categorization of URLs [3], search result evaluation [2], and evaluation of competing translations [20]. In fact, for all these strategies (which correspond to the triangular strategy described earlier), we can replace them with an equivalent rectangular strategy with much lower cost and the same answers. If the ladder or linear algorithms are used, the cost can be reduced even further.

Statistical Hypothesis Testing: Our problem is also related to the field of Statistical Hypothesis Testing [19]. This field is concerned with the problem of trying to estimate whether a certain hypothesis is true or not given the observed data. One such method of estimation is to compute the LR (Likelihood Ratio), i.e., the ratio of the probability that a given hypothesis is true given the data to the probability that an alternative hypothesis is true given the same data. Subsequently the LR is checked to see if it is statistically significant. In our case, our two hypotheses (for a given data item) are simply whether or not the item satisfies the filter. Unlike typical applications of hypothesis testing, where the goal is to estimate the parameters of some distribution, here the distribution is provided to us (i.e., that the item satisfies the filter with probability s). For Problem 2, in fact, our algorithm computes the LR to see if it is greater than the threshold (τ) for all reachable points. However, the hypothesis testing techniques do not help us address the problem of minimizing overall cost while testing a large set of data items.

9. CONCLUSION

We studied the problem of optimizing filtering of data using humans. Our optimal and heuristic algorithms efficiently find filtering strategies that result in significant cost savings relative to commonly-used strategies in crowdsourcing applications.

We focused our presentation primarily on the single-filter problem. Our current solution to the multiple-filters problem is optimal, but has a large blow-up in representation; we hope to find a more compact representation. Other future work includes incorporating human accuracy, handling correlations between filters in the multiple-filters case, extending our techniques for the categorization and clustering problems, and attempting to resolve the open question of whether shapes are optimal for deterministic strategies.

10. REFERENCES

[1] Mechanical Turk. *http://mturk.com*.

[2] Omar Alonso, Daniel E. Rose, and Benjamin Stewart. Crowdsourcing for relevance evaluation. *SIGIR Forum*, 42, 2008.

[3] E. Bakshy et. al. Everyone's an influencer: quantifying influence on twitter. In *WSDM*, 2011.

[4] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. Crowddb: answering queries with crowdsourcing. In *SIGMOD*, 2011.

[5] G. Little et. al. Turkit: tools for iterative tasks on mechanical turk. In *HCOMP*, 2009.

[6] J. Whitehill et. al. Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. In *NIPS*. 2009.

[7] A. Marcus, E. Wu, D. Karger, S. Madden, and R. Miller. Crowdsourced databases: Query processing with people. In *CIDR*, 2011.

[8] A. Marcus, E. Wu, D. Karger, S. Madden, and R. Miller. Demonstration of qurk: a query processor for human operators. In *SIGMOD*, 2011.

[9] Robert McCann, Warren Shen, and AnHai Doan. Matching schemas in online communities: A web 2.0 approach. In *ICDE '08*.

[10] P. Donmez et. al. Efficiently learning the accuracy of labeling sources for selective sampling. In *KDD*, 2009.

[11] P. Perona P. Welinder. Online crowdsourcing: rating annotators and obtaining cost-effective labels. In *CVPR*, 2010.

[12] A. Parameswaran, A. Das Sarma, H. Garcia-Molina, N. Polyzotis, and J. Widom. Human-assisted graph search: it's okay to ask questions. *Proc. VLDB Endow.*, 4:267–278, February 2011.

[13] A. Parameswaran and N. Polyzotis. Answering queries using humans, algorithms and databases. In *CIDR*, 2011.

[14] Alexander J. Quinn and Benjamin B. Bederson. Human computation: a survey and taxonomy of a growing field. In *CHI*, 2011.

[15] R. Snow et. al. Cheap and fast—but is it good?: evaluating non-expert annotations for natural language tasks. In *EMNLP*, 2008.

[16] Burr Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.

[17] V. S. Sheng, F. Provost, and P. Ipeirotis. Get another label? improving data quality and data mining using multiple, noisy labelers. In *SIGKDD*, pages 614–622, 2008.

[18] V. Raykar et. al. Supervised learning from multiple experts: whom to trust when everyone lies a bit. In *ICML*, 2009.

[19] Larry Wasserman. *All of Statistics*. Springer, 2003.

[20] Omar F. Zaidan and Chris Callison-Burch. Feasibility of human-in-the-loop minimum error rate training. In *EMNLP*, 2009.

APPENDIX

A. CONVERSION ALGORITHM

Our algorithm proceeds in two steps: First, we prune unreachable regions. Second, we prune redundant regions.

Step 1: Pruning Unreachable Regions: We define *y-path* to be the sequence of lines leading to the decision point from the point on the y axis in the shape (and including the decision point), and *x-path* to be the sequence of lines leading to the decision point from the point on the x axis in the shape (and including the decision point). We begin by pruning some of the the unreachable portions from the shape. Notice that if the *x-path* ever goes "down", we can instead add a segment that goes "right" until we once again intersect the shape. Additionally, if the *y-path* ever goes "left" we can instead add a segment that goes upwards until we once again intersect the shape. (The decision point may need to be moved if it is unreachable.) After this procedure, we may assume that the *x-path* now has segments that go right, upwards or to the left, while the *y-path* now has segments that go right, upwards and downwards. (Essentially all that we are saying is that for every shape with these unreachable portions, there is another shape without it, with same cost.) This also means that for a given y value there is a single point on the *x-path*, and for a given x value, there is a single point on the *y-path*.

Step 2: Pruning Redundant Regions: We now convert both *x-path* and *y-path* into ladders. We describe our algorithm for the *x-path*, the algorithm for the *y-path* follows similarly. (As we describe the algorithm, we also provide an informal explanation as to why it works.) The algorithm is essentially a scan along the x axis that incrementally builds the ladder.

We begin at $x = 0$, and scan the points corresponding to the *x-path* for the given x coordinate. For some $x = x_i$, we may find that there are some points along the *x-path* that have the x coordinate set to x_i. Find one such point which has the largest such y (say y_i). Now, we add the portion $(x_i, 0) - (x_i, y_i)$ to the ladder. We can do this because no matter what answers we get to the questions to the right of $(x_i, 0)$, with a y coordinate less than y_i, we will always end at a Fail terminating point. Now, we ignore the *x-path* portion below $y = y_i$. (We can effectively assume that we moved the x axis to $y = y_i$ (and the origin to (x_i, y_i)). We now repeat the same procedure. Let us say the next x coordinate for which we find an *x-path* point is $x = x_j$. We add $(x_i, y_i) - (x_j, y_i)$ to the ladder. Let the point on *x-path* with the largest y value at x_j be y_j. We then add $(x_j, y_i) - (x_j, y_j)$ to the ladder. Subsequently, we ignore all *x-path* points below y_j.

In other words, for each x value, we always add a segment that connects the ladder to the ladder until $x - 1$, and optionally a segment that goes from a smaller y value to a larger y value. We keep building the ladder until we hit the decision point. Note that the decision point is the point on the *x-path* with the largest y coordinate, thus we will always hit it. In this manner, we maintain the invariant that all points to the right of the ladder being constructed are all points on *x-path*, and not on *y-path*. (Note that there cannot be any points on the *y-path* to the right of this ladder apart from the decision point because otherwise we will violate the property that the shape has no unreachable regions.) Thus, we ensure that if we ever reach a point on the lower ladder, we are sure to end at a *x-path* point and not an *y-path* point.

Similarly, we build the upper ladder corresponding to the *y-path*. Here the invariant being maintained is that all points above this ladder must be *y-path* points. The decision point is then the point on the *y-path* that has the largest x coordinate. The two ladders meet at the decision point.